Decision Procedures for Separation Logic

Alessio Mansutti

Barbizon 2018

Program verification with Hoare calculus

Hoare calculus is based on proof rules manipulating Hoare triples.

$$\{P\}\ C\ \{Q\}$$

where

- C is a program
- P and Q are assertions in some logical language.

Any (memory) state that satisfies P will satisfy Q after being modified by C.

Programming languages with pointers

The so-called **frame rule**

$$\frac{\{P\}\ C\ \{Q\}}{\{F \wedge P\}\ C\ \{F \wedge Q\}}$$

is generally not valid: it fails if C manipulates pointers.

Programming languages with pointers

The so-called **frame rule**

$$\frac{\{P\}\ C\ \{Q\}}{\{F \wedge P\}\ C\ \{F \wedge Q\}}$$

is generally not valid: it fails if C manipulates pointers.

Example:

$$\frac{\{\exists u.x \mapsto u\}\ [x] \leftarrow 4\ \{x \mapsto 4\}}{\{y \mapsto 3\ \wedge\ \exists u.x \mapsto u\}\ [x] \leftarrow 4\ \{y \mapsto 3\ \wedge\ x \mapsto 4\}}$$

not true if x and y are in aliasing.

Reynolds'02: Separation logic

Separation logic add the notion of **separation** ($*$) of a state, so that the frame rule

$$\frac{\{P\}\ C\ \{Q\} \quad \mathsf{modv}(C) \cap \mathsf{fv}(F) = \emptyset}{\{F * P\}\ C\ \{F * Q\}}$$

is valid.

Reynolds'02: Separation logic

Separation logic add the notion of **separation** ($*$) of a state, so that the frame rule

$$\frac{\{P\}\ C\ \{Q\} \quad \mathsf{modv}(C) \cap \mathsf{fv}(F) = \emptyset}{\{F * P\}\ C\ \{F * Q\}}$$

is valid.

Automatic Verifiers: Infer, SLAyer, Predator (all 2011).

Semi-automatic Verifiers: Smallfoot (2004), Verifast (2008).

Why we need decision procedures for SL?

- Many tools support fragments of Separation Logic as an assertion language.
- Growing demand to consider more powerful extensions:
  - inductive predicates;
  - magic wand operator $-\!\!*$;
  - closure under boolean connectives.
- Deciding satisfiability/validity/entailment is needed.

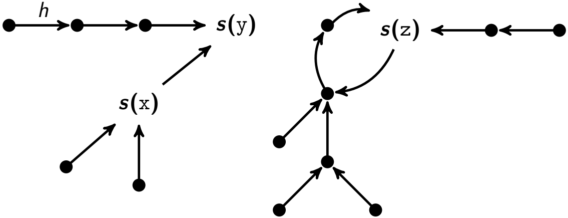
$$\frac{P \implies P' \qquad \{P'\}\ C\ \{Q'\} \qquad Q' \implies Q}{\{P\}\ C\ \{Q\}} \text{ consequence rule}$$

Memory states with one record field

Separation Logic is interpreted over **memory states** (s, h) where:

- $s : \mathtt{VAR} \rightarrow \mathtt{LOC}$ is called store;
- $h : \mathtt{LOC} \rightarrow_{\mathsf{fin}} \mathtt{LOC}$ is called heap.

where $\mathtt{VAR} = \{\mathrm{x}, \mathrm{y}, \mathrm{z}, \dots\}$ set of (program) variables;
$\mathtt{LOC}$ set of locations (typically $\mathtt{LOC} \cong \mathbb{N} \cong \mathtt{VAR}$).



Propositional Separation Logic $\mathtt{SL}(*, -\!\!*)$

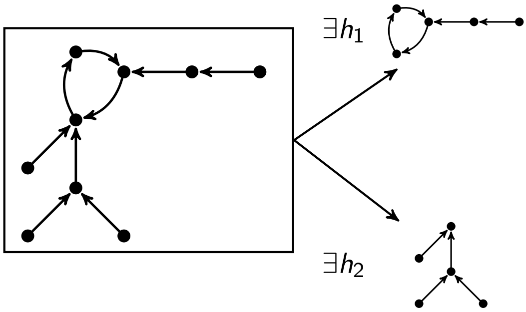
$$\varphi := \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathtt{x} = \mathtt{y} \mid \mathtt{emp} \mid \mathtt{x} \hookrightarrow \mathtt{y} \mid \varphi_1 * \varphi_2 \mid \varphi_1 -\!\!* \varphi_2$$

Semantics

- standard for $\wedge$ and $\neg$;
- $(s, h) \models \mathtt{x} = \mathtt{y} \iff s(\mathtt{x}) = s(\mathtt{y})$
- $(s, h) \models \mathtt{emp} \iff \mathrm{dom}(h) = \emptyset$
- $(s, h) \models \mathtt{x} \hookrightarrow \mathtt{y} \iff h(s(\mathtt{x})) = s(\mathtt{y})$

Separating conjunction ($*$)

$(s, h) \models \varphi_1 * \varphi_2$ if and only if



$(s, h_1) \models \varphi_1$

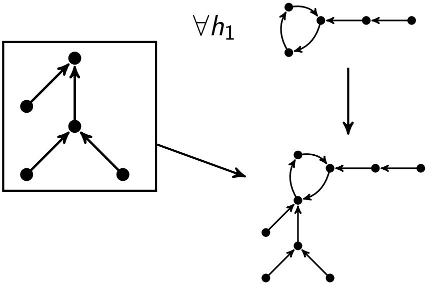
and

$(s, h_2) \models \varphi_2$

There is a way to split the heap into two so that, together with the store, one part satisfies φ_1 and the other satisfies φ_2.

Separating implication ($-\!*$)

$(s, h) \models \varphi_1 -\!* \varphi_2$ if and only if



$\mathrm{dom}(h) \cap \mathrm{dom}(h_1) = \emptyset$
$(s, h_1) \models \varphi_1$

$$\Downarrow$$

$(s, h + h_1) \models \varphi_2$

Whenever a (disjoint) heap that, together with the store, satisfies φ_1 is added, the resulting memory state satisfies φ_2.

Symbolic Heap Fragment (SHF)

$$\Sigma := \texttt{emp} \;\mid\; x \mapsto y \;\mid\; \texttt{ls}(x, y) \;\mid\; \Sigma * \Sigma$$

$$\Pi := x = y \;\mid\; x \neq y \;\mid\; \Pi \wedge \Pi$$

$$\varphi := \Sigma \wedge \Pi$$

- standard fragment in automated tools;
- satisfiability/entailment in $\mathrm{PT{\scriptsize IME}}$;
- boolean combination of SHF is NP-complete;

Extension: $\mathtt{SL}(*, \mathrel{-\!\!*})$ + list segment predicate ($\mathtt{ls}$)

$(s, h) \models \mathtt{ls}(\mathrm{x}, \mathrm{y})$ if and only if

$s(\mathrm{x})$ reaches $s(\mathrm{y})$ and all elements in $\mathrm{dom}(h)$ are necessary for this to hold.

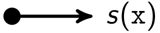
Note: $\mathtt{SL}(*, \mathrel{-\!\!*})$ is already $\mathrm{PS{\scriptsize PACE}}$-complete.

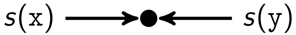
Results (FOSSACS'18)

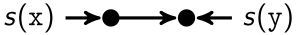
- The satisfiability problem for $\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ is undecidable.
- Several variants of $\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ are also concluded undecidable.
- The satisfiability problem for $\mathtt{SL}(*, \mathtt{ls})$ (i.e. $\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ without $\mathrel{-\!\!*}$) is $\mathrm{PS}\textsc{pace}$-complete.
- The satisfiability problem for Boolean combinations of formulae in $\mathtt{SL}(*, \mathtt{ls}) \cup \mathtt{SL}(*, \mathrel{-\!\!*})$ is $\mathrm{PS}\textsc{pace}$-complete.

Undecidability of SL($*$,$-\!\!*$,$\mathtt{ls}$)

As soon as we add to SL($*$,$-\!\!*$) predicates so that it can express

- $\mathtt{alloc}^{-1}(\mathtt{x}) \iff$ 

- $n(\mathtt{x}) = n(\mathtt{y}) \iff$ 

- $n(\mathtt{x}) \hookrightarrow n(\mathtt{y}) \iff$ 

we obtain a logic with undecidable satisfiabilty problem.

For example:

- SL($*$, $-\!\!*$) + $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 2$ and $\mathtt{reach}(\mathtt{x}, \mathtt{y}) = 3$;
- SL($*$, $-\!\!*$, $\mathtt{ls}$).

Reduction of First-order $\mathtt{SL}(\mathrel{-\!\!*})$ to $\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$

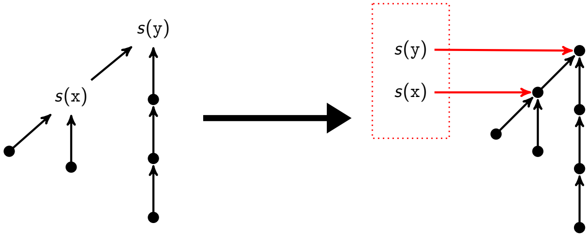
- We consider the first-order extension of $\mathsf{SL}(\mathrel{-\!\!*})$

$$(s, h) \models \forall \mathtt{x}.\varphi \iff \text{for all } \ell \in \mathtt{LOC}, (s[\mathtt{x} \leftarrow \ell], h) \models \varphi$$

- The satisfiability problem for First-order $\mathtt{SL}(\mathrel{-\!\!*})$ is undecidable. [IC, 2012].

- Idea for the translation: use the heap to mimic the store.

Heaps simulate stores



- Given $\mathtt{V} \subseteq_{\mathsf{fin}} \mathtt{VAR}$, take $s|_{\mathtt{V}} + h : \mathtt{VAR} + \mathtt{LOC} \rightarrow_{\mathsf{fin}} \mathtt{LOC}$ and translate it inside the heap domain $[\mathtt{LOC} \rightarrow_{\mathsf{fin}} \mathtt{LOC}]$;

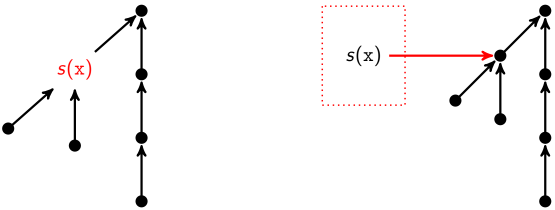
- A finite set of locations is used to simulate a finite portion of the store, effectively splitting the domain $\mathtt{LOC}$.

Undecidability – Some bits of the translation

- $\texttt{translation}_{\texttt{V}}(\texttt{x} = \texttt{y}) \stackrel{\text{def}}{=} n(\texttt{x}) = n(\texttt{y})$;
- $\texttt{translation}_{\texttt{V}}(\texttt{x} \hookrightarrow \texttt{y}) \stackrel{\text{def}}{=} n(\texttt{x}) \hookrightarrow n(\texttt{y})$;
- $\texttt{translation}_{\texttt{V}}(\varphi_1 \mathrel{-\!\!*} \varphi_2) \stackrel{\text{def}}{=}$ too long for a slide;

Universal quantifier – $\forall \texttt{x}.\varphi$

$$(\texttt{alloc}(\texttt{x}) \land \texttt{size} = 1) \mathrel{-\!\!*} (\texttt{safe}(\texttt{V}) \implies \texttt{translation}_{\texttt{V}}(\varphi))$$



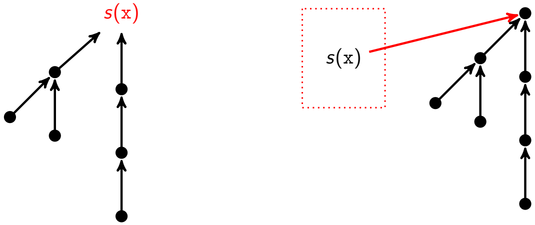
Where $\texttt{safe}(\texttt{V})$ states the sanity conditions to encode the store.

Undecidability – Some bits of the translation

- $\mathtt{translation_V}(\mathtt{x} = \mathtt{y}) \stackrel{\text{def}}{=} n(\mathtt{x}) = n(\mathtt{y})$;
- $\mathtt{translation_V}(\mathtt{x} \hookrightarrow \mathtt{y}) \stackrel{\text{def}}{=} n(\mathtt{x}) \hookrightarrow n(\mathtt{y})$;
- $\mathtt{translation_V}(\varphi_1 \mathrel{-\!\!*} \varphi_2) \stackrel{\text{def}}{=}$ too long for a slide;

Universal quantifier – $\forall \mathtt{x}.\varphi$

$$(\mathtt{alloc}(\mathtt{x}) \land \mathtt{size} = 1) \mathrel{-\!\!*} (\mathtt{safe}(\mathtt{V}) \implies \mathtt{translation_V}(\varphi))$$



Where $\mathtt{safe}(\mathtt{V})$ states the sanity conditions to encode the store.

Deciding $\mathtt{SL}(*, \mathtt{ls})$ thanks to the test formulae approach

- Define sets $\mathrm{Test}_{\mathcal{X}}(n)$ that internalise the role of $*$;
- $*$ elimination: show that each formula of $\mathtt{SL}(*, \mathtt{ls})$ is captured by a boolean combination of test formulae;
- Show a small-model property for the logic of test formulae.

Open problem: to generalise this approach
- identify sufficient conditions on test formulae to have $*$ elimination;
- handle multiple families of test formulae;

$*$ elimination (winning strategy for Duplicator)

For every

- $(s, h) \approx_n (s', h')$;
- $n_1, n_2 \in \mathbb{N}^+$ such that $n = n_1 + n_2$;
- h_1, h_2 disjoint heaps such that $h_1 + h_2 = h$

there are two disjoint heaps h'_1 and h'_2 such that

- $h'_1 + h'_2 = h'$;
- $(s, h_1) \approx_{n_1} (s', h'_1)$ and $(s, h_2) \approx_{n_2} (s', h'_2)$.

Toy Test Formulae $\mathtt{Test}_{\mathcal{X}}(n)$

- $(s,h) \models \#\mathtt{loops}(\beta) \geq \beta' \iff$ the number of loops of size $\beta \leq \mathcal{G}(n)$ is at least β';
- $(s,h) \models \#\mathtt{loops}^{\uparrow} \geq \beta' \iff$ there are at least β' loops of size at least $\mathcal{G}(n) + 1$;
- $(s,h) \models \mathtt{garbage} \geq \beta \iff$ the number of locations not in a loop is at least β

where $\beta \in [1, \mathcal{G}(n)]$ and $\beta' \in [1, \mathcal{L}(n)]$.

Note: these formulae induce a partition on h.

$*$ elimination

Let $(s, h) \approx_n (s', h')$ and let $n_1, n_2 \in \mathbb{N}^+$ such that $n = n_1 + n_2$.
For every h_1, h_2 disjoint heaps such that $h_1 + h_2 = h$...

Bound on $\mathtt{garbage} \geq \beta$ formulae

Given $h = h_1 + h_2$, every location not in a loop of h cannot be in a loop in h_1 or h_2. Then the bound $\mathcal{G}(n)$ must satisfy

$$\mathcal{G}(n) \geq \max_{\substack{n_1, n_2 \in \mathbb{N}^+ \\ n_1 + n_2 = n}} (\mathcal{G}(n_1) + \mathcal{G}(n_2))$$

Bound on $\#\texttt{loops}$ formulae

We consider $\#\texttt{loops}(2) \geq \beta'$ (other cases are similar).
Take $h = h_1 + h_2$. Given a loop of size 2 in h, we identify three cases

- both locations of the loop are assigned to h_1;
- both locations of the loop are assigned to h_2;
- one location of the loop is assigned to h_1 and the other is assigned to h_2.

Then, we search for a bound $\mathcal{L}(n)$ on β' such that

$$\mathcal{L}(n) \geq \max_{\substack{n_1, n_2 \in \mathbb{N}^+ \\ n_1 + n_2 = n}} (\mathcal{L}(n_1) + \mathcal{L}(n_2) + \mathcal{G}(n_1) + \mathcal{G}(n_2))$$

Toy Test Formulae

We have the inequalities

$$\mathcal{G}(1) \geq 1 \qquad \mathcal{G}(n) \geq \max_{\substack{n_1,n_2\in\mathbb{N}^+ \\ n_1+n_2=n}} (\mathcal{G}(n_1) + \mathcal{G}(n_2))$$

$$\mathcal{L}(1) \geq 1 \qquad \mathcal{L}(n) \geq \max_{\substack{n_1,n_2\in\mathbb{N}^+ \\ n_1+n_2=n}} (\mathcal{L}(n_1) + \mathcal{L}(n_2) + \mathcal{G}(n_1) + \mathcal{G}(n_2))$$

Which admit $\mathcal{G}(n) = n$ and $\mathcal{L}(n) = \frac{1}{2}n(n+3) - 1$ as a solution.

For the family $\text{Test}_{\mathcal{X}}(n)$

$$\left\{ \begin{array}{l} \#\texttt{loops}(\beta) \geq \beta',\ \#\texttt{loops}^{\uparrow} \geq \beta', \\ \texttt{garbage} \geq \beta \end{array} \middle| \begin{array}{r} \beta \in [1, n] \\ \beta' \in \left[1, \dfrac{1}{2}n(n+3) - 1\right] \end{array} \right\}$$

we have $*$ elimination.

Test formulae approach (after $*$ elimination)

Suppose we have a family of test formulae $\mathsf{Test}_{\mathcal{X}}(n)$, for all $n \in \mathbb{N}$, such that

- captures the atomic predicates of $\mathtt{SL}(*, \mathtt{ls})$;
- satisfies the $*$ elimination lemma.

Then, let $n \geq |\varphi|$ and $\mathsf{var}(\varphi) \subseteq \mathcal{X}$.

- If $(s, h) \approx_n (s', h')$ then we have $(s, h) \models \varphi$ iff $(s', h') \models \varphi$.
- φ is logically equivalent to a Boolean combination of test formulae from $\mathsf{Test}_{\mathcal{X}}(n)$.

Small model property for boolean combination of $\mathsf{Test}_{\mathcal{X}}(n)$ formulae implies small model property for $\mathtt{SL}(*, \mathtt{ls})$.

Extending FOSSACS paper: $1\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$

$\mathtt{SL}(*, \mathrel{-\!\!*}, \mathtt{ls})$ with one quantified variable u, i.e.

$$(s, h) \models \forall u.\varphi \iff \text{for all } \ell \in \mathtt{LOC}, (s[u \leftarrow \ell], h) \models \varphi$$

Has PSPACE-complete satisfiability problem when $\mathtt{ls}(x, y)$ is constrained so that

- it does not occur on the right side of $\mathrel{-\!\!*}$;
- if $x = u$ then also $y = u$.

Without the first condition: undecidable.
Without the second condition: TOWER-hard.

Proof using two families of test formulae.

Two families of test formulae

$$\Omega := \ldots \mid \exists u.\Omega \mid \Omega_1 * \Omega_2 \mid \Pi \twoheadrightarrow \Omega$$
$$\Pi := \ldots \mid \texttt{reach}^+(x, e) \mid \texttt{reach}^+(u, u) \mid \exists u.\Pi \mid \Pi_1 * \Pi_2 \mid \Pi \twoheadrightarrow \Omega$$

- Separately define test formulae for Ω and Π;
- $*$ elimination and quantifier elimination for both Ω and Π;
- Show that test formulae of Π can express test formulae of Ω. Then, prove $\twoheadrightarrow$ elimination.
- Show small-model property for the logic of test formulae for Π.

Fragment of $\texttt{1SL}(*, \mathrel{-\!\!*}, \texttt{ls})$

- It subsumes other PSPACE-complete fragments of Separation Logic known in the litterature;
- Weakening one of the two conditions most likely makes the problem escape PSPACE.

Also, first PSPACE fragment of Separation Logic that can check

- **garbage freedom**: every model satisfying φ has every memory cell reachable from a program variable occurring in φ.
- **acyclicity**: every model satisfying φ is without loops.

Ongoing work

- Generalising the Test Formulae approach.

Logics

- SPIN'14: Existential fragment of Separation Logic;
- Separation Logic with Inductive Predicates or data values.

Verification

- (Bi-)abduction / Concurrency for SL with reachability;
- IJCAR'18 : Fragment of SL with data values in SMT solver;